

ASP: una *skill* operativa de pre-mortem para agentes de programación con LLM

Un informe de experiencia

Carlos Ulisses Flores

Codex Hash Research Laboratory

ORCID: 0000-0002-6034-7765

Codex Hash Research Report 2026-01

Fecha: 17 de noviembre de 2026

Resumen

Los asistentes de código basados en LLM suelen producir código *localmente correcto* que omite modos de fallo que un ingeniero sénior plantearía en un *pre-mortem*, y la producción falla de formas costosas. Llamamos *shadow points* a esos elementos ausentes. Este informe describe Anticipating Shadow Points (ASP), un protocolo de planificación de 13 fases distribuido como *skill* de Claude Code que integra el *pre-mortem* de Klein (2007) con la taxonomía de fallos multiagente de Cemri et al. (2025), revisa el plan con un validador ajeno al contexto del planificador, según la literatura sobre límites de la autocorrección (Huang et al., 2024; Tyen et al., 2024), y, opcionalmente, entrega la ejecución a un subproceso hijo con contrato en archivos. Documentamos el diseño, su despliegue como *plugin* de código abierto y observaciones preliminares de cinco evaluaciones de ingeniería, leídas por el autor sin cegamiento ni transcripciones conservadas. Un estudio prerregistrado midió el protocolo en seis repositorios públicos: 90 ejecuciones con Claude Opus 5.5, riesgos de referencia enumerados a ciegas por tres modelos de otros proveedores y dos jueces ciegos. ASP cubrió el 48% de los riesgos de referencia, frente al 66% de Claude Code simple instruido a enumerar, antes de planificar, lo que puede salir mal. Quedó detrás en las seis tareas: sin beneficio medible según la regla prefijada. Esa petición elevó la cobertura sobre la petición simple de planificación del 56% al 66% ($p = ,001$), una comparación secundaria prerregistrada que un estudio confirmatorio, presentado aparte, ya probó como pregunta principal.

Palabras clave: agentes de programación con LLM; pre-mortem; modos de fallo multiagente; Claude Code; protocolos operativos; prerregistro

Abstract

LLM coding assistants often produce *locally correct* code that omits failure modes a senior engineer would surface in a pre-mortem, and production then breaks in ways that are expensive to debug. We call these missing items *shadow points*. This experience report describes Anticipating Shadow Points (ASP), a 13-phase planning protocol shipped as a Claude Code skill that integrates Klein's (2007) pre-mortem with the multi-agent failure taxonomy of Cemri et al. (2025), checks the plan with a validator that does not share the planner's context, following the self-correction-limit literature (Huang et al., 2024; Tyen et al., 2024), and, in an opt-in mode, hands execution to a child subprocess with a file-based contract. We document the design, its deployment as an open-source plugin, and preliminary observations from five engineering evaluations, read by the author without blinding and with transcripts not preserved. A pre-registered study then measured the protocol on six public repositories: 90 trials on Claude Opus 5.5, reference risks listed blind by three models of other vendors, and two blind judges. ASP covered 48% of the reference risks against 66% for plain Claude Code asked to list the ways the change could go wrong before planning. It was behind on all six tasks: no measurable benefit under the rule fixed in advance. That single request raised coverage over a plain planning request from 56% to 66% ($p = .001$), a pre-registered secondary comparison that a confirmatory study, reported separately, has since tested as its primary question.

Keywords: LLM coding agents; pre-mortem; multi-agent failure modes; Claude Code; operational protocols; pre-registration

Índice

1. Introducción 5
1.1. Contribuciones 7
2. Antecedentes 8
2.1. Pre-mortem y retrospectiva prospectiva 8
2.2. Modos de fallo multiagente (MAST) 9
2.3. Límites de la autocorrección y aislamiento del validador 10
2.4. El trabajo relacionado más cercano: PreFlect 11
2.5. Linaje clásico de la ingeniería de software y de la calidad 12
3. El protocolo ASP 17
3.1. Visión general 17
3.2. Posicionamiento arquitectónico 18
3.3. Iron Laws 20
3.4. El contrato micro-TODO 21
4. Despliegue operativo 21
5. Observaciones preliminares 23
5.1. El conjunto de evaluaciones 23
5.2. Lo que se observó en despachos reales (autor, no ciego) 24
5.3. Lo que el adjudicador determinista muestra sobre sí mismo 25
5.4. El hallazgo sobre el código de salida de claude -p 28
5.5. Una medición prerregistrada (2026-11) 30
6. Hipótesis de diseño 33
7. Amenazas a la validez 35
8. Limitaciones y trabajo futuro 37
9. Conclusión 38
10. Declaraciones 40
Agradecimientos 41
Referencias 42
Apéndice A — Estado de la verificación de citas 48
Apéndice B — Conjunto de evaluaciones y reproducibilidad 50
Nota Técnica 53
Sobre el autor 55

1. Introducción

En la práctica del autor, el trabajo rutinario de ingeniería —añadir una columna a una base de datos de producción, desplegar una función que llama a una API externa, refactorizar una utilidad usada en treinta archivos— se encarga ahora a menudo a asistentes de programación con IA. Cuando el asistente acierta, el día del ingeniero queda libre. Cuando se equivoca de forma *sutil* —escribiendo código que compila, pasa sus pruebas y se despliega, pero omite en silencio un paso que un ingeniero sénior habría señalado de antemano—, la producción se rompe de formas costosas de depurar. La ciencia de la decisión tiene una técnica para justo esa brecha, el pre-mortem (Klein, 2007), y la investigación sobre agentes dispone ahora de un catálogo de cómo fallan los sistemas multiagente basados en LLM (Cemri et al., 2025). Nuestra tesis es que ambos pertenecen juntos en tiempo de ejecución: antes de escribir código, debe obligarse al agente a ejecutar un pre-mortem frente a una taxonomía de fallos fija, y su plan debe ser revisado por un validador que no pueda ver el razonamiento del planificador.

El modo de fallo de los agentes de programación con LLM que motivó ASP no es el código *malo*. Es el código *incompleto*: el agente hace lo que se le pidió, pero no lo que alguien con un contexto profundo del dominio se habría preguntado *antes de empezar*. El retraso de réplica durante una migración de esquema. La cabecera CORS de la función en el edge, que indica al navegador qué otros sitios pueden llamarla. La política de seguridad a nivel de fila (RLS), que limita las filas que cada usuario puede leer y hay que actualizar para que coincida con la nueva columna. Llamamos **shadow points** a estos elementos ausentes: modos de fallo evidentes para un profesional al que ya le han quemado, e invisibles para un agente al que no.

Dos corrientes de la literatura convergen en esta brecha. La primera, en la ciencia de la decisión, es el **pre-mortem**: un ejercicio estructurado en el que un equipo imagina que un proyecto ya ha

fracasado y enumera las causas antes de actuar. La técnica fue operacionalizada para equipos de proyecto por Klein (2007) y se apoya en el trabajo empírico anterior de Mitchell, Russo y Pennington (1989) sobre la retrospectiva prospectiva (*prospective hindsight*); Klein (2007) la resume como un aumento de alrededor del 30% en la capacidad de identificar las razones de un resultado futuro, una cifra que no pudimos rederivar a partir del artículo de 1989 (Apéndice A). La segunda, en la investigación sobre agentes, es la reciente caracterización de los modos de fallo multiagente de Cemri et al. (2025), quienes derivaron una taxonomía de 14 modos, la Multi-Agent System Failure Taxonomy (MAST), a partir de 150 trazas semilla de sistemas multiagente basados en LLM (acuerdo entre anotadores $\kappa = ,88$) y publicaron un corpus de más de 1.600 trazas anotadas. La crítica del plan de un agente antes de que actúe ya existe (Sección 2.4); lo que no encontramos son las dos corrientes integradas como una **disciplina de tiempo de ejecución**, con artefactos con nombre y un validador que no comparte el contexto del planificador, que se pueda obligar a seguir a un agente de programación en toda tarea no trivial.

Este artículo informa sobre **Anticipating Shadow Points (ASP)**, un protocolo de planificación de 13 fases distribuido como *skill* de Claude Code que combina el razonamiento de pre-mortem con la lista de verificación de 14 modos de MAST, separa la planificación de la validación bajo aislamiento de *prompts* y, en su modo opcional --goal, entrega la ejecución a un subproceso hijo con un contrato basado en archivos (por defecto, la ejecución permanece en la sesión del usuario). Desplegamos la *skill*, la distribuimos por tres vías de instalación (*marketplace* de *plugins*, modo desarrollador, *script* independiente), la sometimos a cinco evaluaciones de ingeniería e informamos del diseño y de lo que las evaluaciones pueden y no pueden mostrar. Estas cinco evaluaciones son preliminares y no medidas: GREEN (el agente con la disciplina de ASP) en cinco tareas y RED (al que se pidió planificar sin ella) en tres, un despacho cada uno, un solo modelo,

leída por el autor sin cegamiento, con las transcripciones en bruto no conservadas. Las informamos como observación cualitativa. Una medición preregistrada llegó después (Sección 5.5): en seis tareas, las fases de planificación de ASP cubrieron menos riesgos de referencia que una petición de una frase. Las cuatro decisiones de diseño que tratamos como hipótesis, y una quinta medida en el artículo complementario, están en la Sección 6.

1.1. Contribuciones

Este informe de experiencia aporta:

1. **El protocolo ASP como diseño documentado** — trece fases, cada una de las cuales emite un artefacto con nombre, organizadas en torno al principio de que ninguna implementación comienza antes de que se hayan sacado a la luz los shadow points.
2. **Doce restricciones operativas que llamamos "Iron Laws"** (Sección 3.3), cada una vinculada a una fuente primaria de la literatura o a una decisión de diseño documentada. Dos tienen una comprobación a nivel de *shell* en la plantilla ejecutora que se le dice al agente que ejecute (la salida JSON de la Iron Law 12, un valor por defecto de la plantilla que el operador puede sobrescribir, y la bandera ASP_IN_GOAL de la Iron Law 9, definida para el hijo para que ASP no vuelva a arrancar dentro de él, cuya comprobación de entrada es un fragmento de Bash que la *skill* indica al agente que ejecute); las otras diez son instrucciones a nivel de *prompt*.
3. **Observaciones preliminares y cualitativas** de cinco evaluaciones de ingeniería, y un adjudicador determinista con procedencia criptográfica, ejercitado sobre *fixtures* construidos (planes de ejemplo que el autor escribió para probarlo) y listo para puntuar transcripciones reales sin depender de la adjudicación del autor (las listas esperadas

siguen siendo del autor, y el *sidecar* opcional de sinónimos no se proporciona para ninguna tarea).

4. **Un hallazgo empírico aparte** sobre la semántica del código de salida de claude -p (Claude Code ejecutado de forma no interactiva: entra un *prompt*, sale un resultado) ante el rechazo cortés de un agente, caracterizado en el artículo complementario Flores (2026b). El hallazgo motivó dos de las Iron Laws, pero merece una caracterización independiente con un N mayor.

Este es un **informe de experiencia** en el sentido de los sistemas, no un artículo de *benchmark*. Documentamos un sistema que construimos, una metodología que desplegamos y lo que observamos; no afirmamos una contribución empírica sólida a la literatura de *benchmarks* de agentes LLM. La Sección 7 enumera en detalle las amenazas a la validez.

2. Antecedentes

Repasamos cuatro líneas de trabajo previo reciente que informan directamente el diseño de ASP (§§2.1–2.4) y luego situamos a ASP dentro de las disciplinas más antiguas de ingeniería de software y de calidad que transpone (§2.5). Mantenemos breve el repaso del trabajo reciente en comparación con los artículos habituales sobre sistemas de agentes, porque la contribución aquí es operativa y el contexto bibliográfico está al servicio de ella.

2.1. Pre-mortem y retrospectiva prospectiva

La técnica del pre-mortem pide a un equipo que imagine, antes de comprometerse, que un proyecto ya ha fracasado de forma catastrófica, y que enumere las causas. La base empírica es Mitchell, Russo y Pennington (1989), quienes mostraron que imaginar un hecho futuro como cierto (en lugar de incierto) produce explicaciones más largas y más episódicas; Klein (2007) resume el efecto

como una mejora de alrededor del 30% en la identificación correcta de las razones de un resultado futuro. Klein (2007) operacionalizó el hallazgo como el "pre-mortem" de equipos de proyecto. Su base experimental es escasa: no encontramos ninguna réplica del efecto de alrededor del 30% que Klein (2007) atribuye a Mitchell et al. (1989), ni ningún metaanálisis de él, en Crossref, arXiv ni en una búsqueda web general (noviembre de 2026). Eso es una ausencia en esos índices, no una prueba de que no exista ninguna, y no dice nada sobre experimentos con otros efectos del pre-mortem. Las normas de evaluación de riesgos catalogan técnicas vecinas —el análisis de escenarios y la técnica estructurada de hipótesis (SWIFT)—, pero el índice de IEC 31010:2019, Anexo B, no menciona el pre-mortem por su nombre. El término *prospective hindsight* también se ha reutilizado para una señal de aprendizaje por refuerzo sin relación (J. Zhang et al., 2026); nosotros lo usamos solo en el sentido de Mitchell et al. La crítica del plan de un agente antes de que actúe ya existe (PreFlect; Wang et al., 2026; §2.4). Lo que no encontramos es la integración que hace ASP: un pre-mortem ejecutado frente a una taxonomía de fallos fija, registrado como artefactos con nombre y revisado por un validador que no comparte el contexto del planificador.

2.2. Modos de fallo multiagente (MAST)

Los agentes LLM intercalan razonamiento y acción en un único bucle (Yao et al., 2023) y se componen cada vez más en flujos de trabajo multiagente cuyos modos de fallo se acumulan. Cemri et al. (2025) presentan **MAST** con **MAST-Data**, un corpus de más de 1.600 trazas anotadas de siete *frameworks* multiagente populares como AG2 (antes AutoGen; Wu et al., 2023). A partir de un análisis riguroso de 150 trazas semilla (acuerdo entre anotadores $\kappa = ,88$), derivaron catorce modos de fallo organizados en tres categorías: especificación y diseño del sistema (5 modos), desalineación entre agentes (6 modos) y verificación y terminación de tareas (3 modos); las tablas de la Sección 3 citan estos modos por sus códigos, de FM-1.1 a FM-3.3. La lista de 14 modos es

el fundamento empírico de la lista de verificación de modos de fallo de ASP: ASP adopta los nombres de los modos literalmente y operacionaliza cada uno en un *prompt* de pre-mortem más una mitigación en sus archivos de *skill* complementarios. Dos trabajos concurrentes amplían la línea de la taxonomía: Zhu et al. (2025) presentan AgentDebug y la AgentErrorTaxonomy (memoria / reflexión / planificación / acción / nivel de sistema), y S. Zhang et al. (2025), un Spotlight de ICML 2025, abre la **atribución automatizada de fallos** con el conjunto de datos Who&When. MAST es, por tanto, una taxonomía entre varias: TRAIL formaliza una taxonomía de errores agénticos sobre 148 trazas anotadas por humanos (Deshpande et al., 2025), AgentRx anota 170 trayectorias fallidas con un paso crítico y una categoría de una taxonomía de teoría fundamentada (Barke et al., 2026), y AgenTracer informa de que los LLM de razonamiento actuales atribuyen los fallos en trazas multiagente con una exactitud generalmente inferior al 10% (G. Zhang et al., 2025). ASP usa MAST como biblioteca de *prompts* para anticipar fallos, no como clasificador de los observados, de modo que la elección de la taxonomía delimita lo que la Fase 2 pregunta y no lo que puede detectar.

2.3. Límites de la autocorrección y aislamiento del validador

El argumento para separar el validador del planificador se apoya en tres trabajos primarios según los cuales los LLM no son fiables al criticarse a sí mismos bajo un contexto compartido. Estos hallazgos motivan, pero no establecen, la hipótesis de que separar el *prompt* del validador de la transcripción del planificador mejora la validación en ASP. Huang et al. (2024, ICLR) muestran que la autocorrección intrínseca sin retroalimentación externa a menudo degrada el rendimiento. Tyen et al. (2024) descomponen la autocorrección en localización y corrección y muestran empíricamente que los LLM no pueden *localizar* sus propios errores de razonamiento pero sí corregirlos si se les da la ubicación desde fuera: *la localización es el cuello de botella*. Zheng et

al. (2023, NeurIPS D&B) documentan que los jueces LLM potentes favorecen en exceso las salidas de modelos de la misma familia (un "self-enhancement bias") incluso al juzgar objetivamente la calidad. Kamoi et al. (2024) repasan esta literatura y concluyen que la autocorrección fiable de los errores de razonamiento requiere retroalimentación o verificadores externos y no el juicio del modelo sin ayuda; los métodos de autorretroalimentación intrínseca como Self-Refine (Madaan et al., 2023) mejoran algunas tareas de generación pero no cierran la brecha de localización que aíslan Tyen et al. (2024). El mecanismo que suponemos, y que no ponemos a prueba aquí (Sección 6), es que un validador que comparte contexto con un planificador hereda la calibración de confianza del planificador sobre el contenido compartido, incluido el contenido erróneo del que el planificador está seguro. El diseño de ASP responde indicando el aislamiento de *prompts* entre planificador y validador (Iron Law 2, más abajo). Kambhampati et al. (2024) sostienen, para la planificación, que los LLM autorregresivos no pueden planificar ni autoverificarse por sí solos, pero pueden servir como fuentes aproximadas de conocimiento dentro de un bucle "LLM-Modulo" con verificadores externos. Las pruebas de aceptación ejecutables de ASP son verificadores de esa clase externa. Su validador no lo es: es un segundo *prompt* al mismo modelo, que separa el contexto pero no el modelo.

2.4. El trabajo relacionado más cercano: PreFlect

El trabajo concurrente más cercano es **PreFlect** (Wang et al., 2026), que de forma similar desplaza el paradigma de la autorreflexión de la corrección a posteriori a la crítica previa a la ejecución de los planes del agente, destilando errores de planificación a partir de trayectorias históricas de agentes y añadiendo un mecanismo de replanificación dinámica. La similitud de alto nivel es significativa: tanto ASP como PreFlect identifican como una limitación que la reflexión

convencional solo ocurra después de la ejecución, y desplazan la crítica *antes* de la acción. ASP se diferencia de PreFlect en cuatro ejes:

1. **Validador aislado por *prompt*.** PreFlect destila los errores en la propia crítica previa a la ejecución del agente; ASP indica un subagente validador de *prompt* separado al que no se da la transcripción del planificador.
2. **Aislamiento a nivel de proceso para el ejecutor.** En su modo opcional --goal, ASP lanza el ejecutor como un subproceso claude -p con un contrato basado en archivos; el artículo de PreFlect no describe una frontera de proceso equivalente.
3. **Restricciones explícitas de tiempo de ejecución.** ASP enuncia doce Iron Laws, dos de ellas protegidas por comprobaciones ejecutables y diez como reglas a nivel de *prompt*; PreFlect se presenta como una metodología y no como un sistema de restricciones.
4. **Un hallazgo empírico a nivel de CLI** que caracteriza la semántica de salida de claude -p ante un rechazo cortés y que motiva dos de las Iron Laws; se informa por separado en Flores (2026b). El fenómeno en sí está documentado por el proveedor y en informes públicos de *issues* (catalogados en Flores, 2026b, §2); la contribución allí es un recuento medido y una propuesta de contrato de terminación tipado, no un descubrimiento.

2.5. Linaje clásico de la ingeniería de software y de la calidad

Las cuatro líneas anteriores sitúan a ASP dentro de la literatura reciente sobre agentes LLM. La maquinaria de ASP es, sin embargo, una transposición de disciplinas de ingeniería de software y de calidad que preceden a los LLM en décadas. Lo nuevo en ASP no son las disciplinas sino su transposición a un entorno de ejecución de agentes no determinista y los mecanismos que las hacen cumplir allí.

Comprobación obligatoria de postcondiciones: ASP y el diseño por contrato (*Design by Contract*). El esquema de micro-TODO de ASP (§3.4) es el diseño por contrato (Meyer, 1992) trasladado de la frontera de la rutina a la frontera de la tarea del agente. El contrato de Meyer reparte obligaciones a través de una frontera llamador/rutina: el llamador debe establecer la PRECONDICIÓN, que la rutina tiene entonces derecho a *asumir* —el principio de no redundancia de Meyer prohíbe que el cuerpo la vuelva a comprobar—, y la rutina debe garantizar la POSTCONDICIÓN en la que se apoya el llamador. De los siete campos del esquema, PRECONDITION, ACTION y POSTCONDITION se corresponden directamente con este contrato, con ACTION como el cuerpo de la rutina que envuelve el par pre/post. Meyer (1992) prevé también la monitorización de aserciones en tiempo de ejecución, postcondiciones incluidas, como una opción elegida en la compilación; por defecto solo se monitorizan las precondiciones. ASP convierte esa opción en regla: como el obligado es un modelo no determinista y no una rutina compilada, la postcondición de cada paso se comprueba con un ACCEPTANCE-TEST externo, una orden ejecutada fuera del modelo. La contrapartida: ASP hace obligatoria la monitorización de las postcondiciones y paga una comprobación en cada paso, en tiempo y en tokens, para obtener evidencia explícita de que el paso se completó, porque nada garantiza que un modelo haya cumplido su contrato. La regla se inspira en el diseño por contrato, pero es distinta del rechazo de Meyer a las comprobaciones redundantes de precondiciones, que se refiere a una rutina que vuelve a comprobar lo que su llamador ya garantizó. El FALSIFICATION-TEST añade la postura popperiana (Popper, 1959/2002) de que una afirmación gana confianza al sobrevivir a un intento de refutación y no solo por una comprobación confirmatoria; funciona como una prueba negativa junto al contrato, no como una cláusula de este. La misma postura produce la disciplina del código de salida de las Iron Laws 11–12. Las Iron Laws, a su vez, se enuncian como el análogo del tercer

elemento del contrato en DbC —el invariante de clase— y no como una pre/postcondición por tarea; diez de ellas son instrucciones que se le dice al agente que mantenga, no comprobaciones que se ejecuten.

Enumeración de modos de fallo. La detección de shadow points (Fase 2) es una enumeración prospectiva de modos de fallo en la tradición del análisis de modos de fallo y efectos (FMEA; MIL-STD-1629A, 1980; IEC 60812:2018). Deliberadamente *no* es un FMEA completo: ASP no realiza ninguna descomposición por componente ni puntuación de prioridad de riesgo (gravedad × ocurrencia × detección), de modo que la deuda es con la disciplina de enumeración prospectiva del FMEA, no con su mecánica ascendente. Tres entradas de la Fase 2 ocupan papeles que conviene separar: el pre-mortem (Klein, 2007; §2.1) aporta la *filosofía* —imaginar el fallo como ya cierto—; la taxonomía de catorce modos de MAST (Cemri et al., 2025; §2.2) aporta la *biblioteca* de modos de fallo; y la enumeración forzada frente a esa biblioteca fija se parece más, estructuralmente, al uso de palabras guía de los estudios de peligros y operabilidad (HAZOP; IEC 61882:2016), en los que una lista fija de palabras orienta la búsqueda de desviaciones, que al FMEA componente por componente. Lo que ASP añade es el sustrato: el "componente" cuyos modos de fallo se enumeran es un plan de LLM; el planificador ejecuta la enumeración (Fase 2), y un validador que no comparte el contexto del planificador revisa la lista resultante (Fase 5; §2.3).

A prueba de errores. Un subconjunto de las Iron Laws (§3.3) son *poka-yoke* (Shingo, 1986), dispositivos que hacen físicamente imposible un error, a nivel de protocolo: la plantilla ejecutora de claude-p-goal-runner.md define ASP_IN_GOAL para el hijo y pasa --output-format json por defecto (Iron Laws 9 y 12); el contrato basado en archivos de la Iron Law 10 es una instrucción. Estas comprobaciones viven en una plantilla Bash que se le dice al agente que ejecute, no en un *hook* (una orden que el propio ejecutor lanza) fuera del control del agente, y no son la imposibilidad

física del *poka-yoke* auténtico —una variable de entorno puede desactivarse o no propagarse—, y las demás leyes son compuertas procedimentales (Iron Laws 1, 2, 4, 5) o reglas de verificación que siguen dependiendo del operador, de las cuales la Iron Law 11 ("nunca confiar en $?", el registro del último código de salida que guarda el *shell*) es el caso más claro: precisamente la dependencia de la vigilancia del operador frente a la cual se define el *poka-yoke*. El préstamo es la *postura* de prevención de errores aplicada a dos comprobaciones a nivel de *shell* en un *script* ejecutor que se le indica al agente que emita, no una afirmación de que ASP elimine la vigilancia del operador.

Verificación y validación independientes. El validador aislado por *prompt* de ASP (Iron Law 2) transpone el movimiento definitorio de la verificación y validación independientes (IV&V; IEEE Std 1012-2016) —un verificador mantenido independiente del desarrollador, en términos de IEEE 1012 técnica, gerencial y financieramente— de la separación organizativa a la separación de *prompts*: el subagente validador no comparte contexto con el planificador al que revisa. La justificación específica de los LLM para ese aislamiento es la literatura sobre autocorrección del §2.3; la *forma* del contramovimiento es la de IV&V.

Vocabulario de la confiabilidad. El comportamiento del código de salida detrás de las Iron Laws 11–12 tiene un nombre exacto en la taxonomía de la confiabilidad: un exit 0 ante un rechazo cortés es un *fallo no señalizado* (Avizienis et al., 2004), ni una caída ni una parada fail-silent (que se detiene sin emitir nada). El término, sus antecedentes y la medición se desarrollan en el artículo complementario (Flores, 2026b); esta sección importa solo el vocabulario y no reabre aquí la cuestión de la novedad.

Garantía empírica. La preferencia por pruebas de aceptación y de falsación frente a códigos de estado en los que se confía tiene una base empírica en la patología adyacente. Yuan et al. (2014) encontraron que una gran mayoría de los fallos catastróficos en sistemas distribuidos intensivos en

datos se remontan a un manejo defectuoso de errores no fatales que *sí* se señalaron. El caso del código de salida de ASP es el complemento —la señal estaba *ausente*—, de modo que ambos no comparten una tesis; la estrecha lección que sí comparten es que un canal de estado no es un sustituto del desenlace, que es lo que existe un ACCEPTANCE-TEST externo para hacer cumplir.

Tabla 1 asigna cada mecanismo de ASP a su ancestro clásico y declara lo que ASP añade.

Tabla 1
Mecanismos de ASP y sus ancestros clásicos

Mecanismo de ASP	**Ancestro clásico**	**Lo que ASP añade (y *no* afirma)**
micro-TODO PRE/ACTION/POST + ACCEPTANCE-TEST	Diseño por contrato (Meyer, 1992)	un obligado no determinista cuya postcondición se comprueba con un comando externo en cada paso, mientras que la monitorización en tiempo de ejecución de Meyer es una opción; no la idea del contrato
micro-TODO FALSIFICATION-TEST	Falsación popperiana (Popper, 1959/2002)	un paso de refutación en la frontera de la tarea del agente; una prueba negativa junto al contrato, no una cláusula de este
Detección de shadow points (Fase 2)	Enumeración al estilo FMEA (MIL-STD-1629A; IEC 60812:2018) + palabras guía de HAZOP, sobre la biblioteca MAST (Cemri et al., 2025)	enumeración sobre un plan de LLM por el planificador, revisada por un validador aislado por *prompt*; sin puntuación de prioridad de riesgo ni descomposición por componente
Iron Laws 9 y 12 como comprobaciones a nivel de *shell*	*Poka-yoke* (Shingo, 1986)	dos comprobaciones en un *script* ejecutor que se le indica al agente que emita; no una imposibilidad física, y la Iron Law 10 es una instrucción
Validador aislado por *prompt* (Iron Law 2)	IV&V (IEEE Std 1012-2016)	independencia organizativa reformulada como aislamiento de *prompts*; no es IV&V
Nunca confiar en $? (Iron Laws 11–12)	Fallo no señalizado (Avizienis et al., 2004); Yuan et al. (2014)	caracterizado en el artículo complementario (Flores, 2026b); no la clase de fallo

Leída en conjunto, la sección hace una afirmación modesta: la maquinaria de ASP es una *integración* de disciplinas establecidas de confiabilidad e ingeniería de la calidad —contrato, enumeración de modos de fallo, IV&V, prevención de errores— en un único protocolo cuya

contribución es operativa y no conceptual. Lo que esa integración añade más allá de sus ancestros es su aplicación en un entorno de ejecución de agentes no determinista: las Iron Laws (dos de ellas protegidas por comprobaciones ejecutables), el aislamiento a nivel de *prompt* y de proceso del §3.2 y el adjudicador determinista reproducible del §5, con el hallazgo sobre el código de salida de claude -p caracterizado por separado en el artículo complementario. Esta es la afirmación de informe de experiencia del §1, no una afirmación de novedad frente a la literatura clásica.

3. El protocolo ASP

3.1. Visión general

ASP es un protocolo de 13 fases que avanza desde la recepción hasta la conformidad de entrega. Cada fase emite un artefacto con nombre (Tabla 2). El principio que organiza las fases es que ninguna implementación comienza antes de que los shadow points se hayan sacado a la luz (Fase 2), revisado por un validador aislado por *prompt* (Fase 5; el modo --quick omite esta revisión, el modo por defecto y el --paranoid la mantienen) y aprobado por el usuario (Fase 7); la ejecución permanece por defecto en la sesión del usuario y, con la *flag* opcional --goal, se delega a un subproceso hijo con un contrato basado en archivos (Fase 9).

Tabla 2
Fases de ASP y los artefactos que emiten. La Fase 0 tiene dos subfases (0a antes de la investigación, 0b después), de modo que trece fases ocupan catorce filas

#	Fase	Artefacto	Fundamento del diseño
0a	Preguntas y respuestas previas a la investigación	bloque intake	Buena práctica de Anthropic *Skills*: salidas intermedias verificables
1	Investigación en paralelo	Resumen de la investigación + bloque de *recall*	Patrón de despacho de subagentes
0b	Preguntas y respuestas posteriores a la investigación	*Intake* refinado (omitible)	Igual que 0a

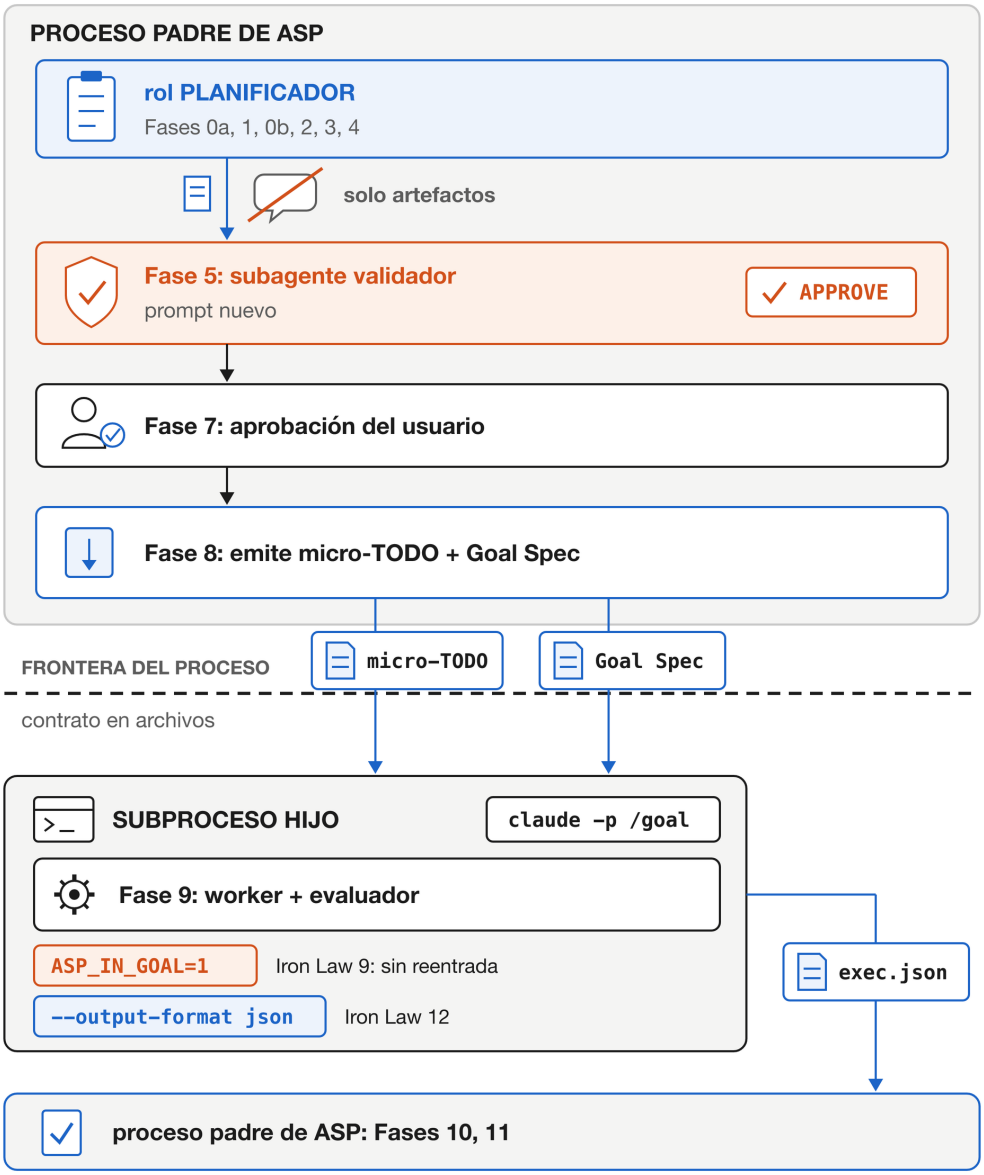
#	Fase	Artefacto	Fundamento del diseño
2	Detección de shadow points	Lista categorizada + mitigaciones	Klein (2007) + Cemri et al. (2025)
3	*Project Charter*	*Charter* completado	Traspaso duradero basado en artefactos
4	Plan macro + registro	Plan macro + entregables D01–D*n*	Plan-and-Act (Erdogan et al., 2025)
5	Validador independiente	APPROVE / REVISE	Huang et al. (2024) + Tyen et al. (2024) + Zheng et al. (2023)
6	Bucle interno (tope=3)	Plan de la revisión final	Iteración acotada; FM-1.5
7	Aprobación del usuario	Go / no-go	Compuerta con humano en el bucle
8	Micro-TODO + Goal Spec	Una tarea TaskCreate (la herramienta de lista de tareas de Claude Code) por paso + Goal Spec (la condición de finalización escrita para el ejecutor)	Traspaso al ejecutor
9	Ejecución, en la sesión o, con la *flag* opcional --goal de ASP, en un subproceso hijo que ejecuta el comando /goal de Claude Code	Tareas marcadas como completadas con evidencia	Aislamiento a nivel de proceso en el modo opcional; véase Flores (2026b)
10	Conformidad por entregable	Cada entregable aceptado o rechazado	Contrato de aceptación
11	Informe de ejecución + memoria	Artefacto de cierre + escritura de memoria	Memoria al estilo Reflexion (Shinn et al., 2023)
12	(Opcional) publicación pública	*Marketplace* de *plugins* + *release* de GitHub	Capa de distribución

3.2. Posicionamiento arquitectónico

ASP separa tres roles de agente con traspasos explícitos solo mediante artefactos (Figura 1).

Figura 1

Separación en tres roles de ASP



Nota. Las fases del planificador se ejecutan en el padre. El validador se ejecuta en el padre pero en un *prompt* de subagente nuevo que recibe solo artefactos, nunca la transcripción del planificador. Con la *flag* opcional --goal, el caso dibujado aquí, el *worker* se ejecuta en un subproceso hijo claude -p con un contrato basado en archivos; por defecto se ejecuta en la sesión del usuario.

3.3. Iron Laws

Enunciamos doce restricciones operativas y las llamamos Iron Laws (Tabla 3). Las Iron Laws 9 y 12 tienen comprobaciones a nivel de *shell* en la plantilla ejecutora y en el fragmento de entrada que se le indica al agente que ejecute; las demás son reglas que la *skill* instruye al agente a seguir. El cumplimiento de ambos tipos no se ha medido. Cada una se vincula a su derivación: una fuente primaria de la literatura, una recomendación documentada de Anthropic o una decisión empírica de diseño.

Tabla 3
Las 12 Iron Laws y su derivación

#	Ley	Derivación
1	Ninguna implementación sin una pasada previa de shadow points	Klein (2007) + Cemri et al. (2025)
2	Ninguna aprobación del plan sin una pasada de validador independiente (*prompt* separado); el modo --quick omite el validador	Huang et al. (2024) + Tyen et al. (2024) + Zheng et al. (2023); mitiga FM-3.3
3	Ningún micro-TODO sin plan macro + *charter* + entregables aprobados por el usuario	Anthropic *Skills*: salidas intermedias verificables
4	Ninguna finalización de paso sin evidencia reciente de pruebas de aceptación	Cemri et al. (2025) FM-3.1 + FM-3.2 + FM-2.6
5	Ningún entregable aceptado sin que coincida el archivo o la salida de evidencia	Cemri et al. (2025) FM-3.1
6	Ningún cierre de proyecto sin informe de ejecución + escritura de memoria	Búfer de memoria de Shinn et al. (2023)
7	Ningún bucle sin tope (tope por defecto=3 rondas de validador)	Cemri et al. (2025) FM-1.5
8	Ninguna ejecución de la Fase 9 sin despacho del evaluador en los puntos de control (salvo con --quick)	Planificador/ejecutor de Erdogan et al. (2025)
9	Ningún ASP reentrante dentro del hijo claude -p /goal (guarda con la variable de entorno ASP_IN_GOAL)	Mitigación de la explosión de costos; decisión de diseño
10	Ningún lanzamiento de claude -p /goal sin contrato basado en archivos	Mitigación de la divergencia de estado entre subprocesos; decisión de diseño
11	Nunca confiar en $? de claude -p para el éxito o fracaso semántico	Observación empírica, caracterizada en Flores (2026b)
12	Toda invocación de claude -p debe usar --output-format json o stream-json	Corolario de la Iron Law 11

3.4. El contrato micro-TODO

En la frontera entre la planificación y la ejecución (Fase 8), ASP emite una lista de tareas contractual. Cada tarea se crea mediante la primitiva TaskCreate con siete campos etiquetados:

- **PRECONDITION** — invariante que debe cumplirse antes del paso.
- **ACTION** — comando exacto o edición literal.
- **POSTCONDITION** — cambio de estado que efectúa el paso.
- **ACCEPTANCE-TEST** — comando y salida esperada que prueban la postcondición.
- **FALSIFICATION-TEST** — comando u observación que probaría que el paso falló.
- **DEPENDS-ON** — identificadores de tareas previas (un grafo acíclico dirigido, o DAG: dependencias sin ciclos).
- **DELIVERABLE-ID** — a qué entregable del registro contribuye este paso.

Este esquema es lo que hace auditable la lista de tareas: ninguna tarea se marca como completada sin evidencia de pruebas de aceptación (Iron Law 4); ningún entregable se da por conforme sin evidencia coincidente (Iron Law 5). El esquema se comparte con el Goal Spec que el subproceso de la Fase 9 consume en el modo opcional --goal.

4. Despliegue operativo

ASP se distribuye como *plugin* de Claude Code por tres vías de instalación, cada una adecuada para un público distinto:

1. ***Marketplace* de *plugins*** (recomendado para usuarios finales):

```
claude plugin marketplace add ulissesflores/anticipating-shadow-points
claude plugin install anticipating-shadow-points@anticipating-shadow-
points
```

Invocación: /anticipating-shadow-points:asp <task> o /asp a secas.

2. **Modo desarrollador** (para colaboradores): claude --plugin-dir ./plugins/anticipating-shadow-points carga la *skill* in situ; útil al iterar sobre la propia *skill*.
3. ***Script* de instalación independiente** (para usuarios que no usan el gestor de *plugins*): ./scripts/install.sh copia la *skill* en el directorio de *skills* de Claude Code del usuario.

El *plugin* usa el formato oficial de manifiesto .claude-plugin/plugin.json de Anthropic (Claude Code 2.1+; Anthropic, 2026a). Vive en una carpeta propia, plugins/anticipating-shadow-points/, que solo contiene Markdown, el icono y la licencia, de modo que la instalación no copia nada más; la raíz del repositorio funciona además como *marketplace* propio mediante .claude-plugin/marketplace.json.

Permisos. Desde la versión 1.0.4 del *plugin*, el ejecutor de la Fase 9 del modo opcional --goal lanza el hijo claude -p **sin ninguna opción de permisos**: el hijo se ejecuta bajo la propia configuración de permisos de Claude Code del usuario, y la *skill* nunca los concede, amplía ni elude. Las versiones anteriores pasaban --permission-mode bypassPermissions; eso se eliminó después de que el directorio de *plugins* de Claude, al revisar el *plugin*, señalara que un *plugin* no debe concederse permisos a sí mismo (CHANGELOG.md, versión 1.0.4). En modo no interactivo, una acción que el usuario no ha permitido de antemano se deniega en lugar de solicitarse; el padre lee el campo permission_denials del resultado JSON del hijo e informa al usuario de cualquier denegación en lugar de reintentar con un permiso más amplio. Esto hace que la ejecución autónoma sea estrictamente más débil que en el despliegue de mayo de 2026, y más segura: la *skill* solo puede hacer lo que el operador ya permitió. Desde la versión 2.0.0, la *skill* además avisa al usuario, antes de lanzar el primer hijo, de que este se ejecuta con los permisos y el consumo del propio usuario,

con el presupuesto y el límite de turnos declarados, y solo lo lanza tras su OK; los archivos de trabajo del hijo van a la carpeta de datos del propio *plugin*, que Claude Code borra al desinstalarlo. La propia *skill* está implementada como una *skill* basada en Markdown que sigue el documento de buenas prácticas de Agent *Skills* de Anthropic (Anthropic, 2026b), con archivos complementarios para la lista de verificación MAST, los *prompts* de pre-mortem por modo y las siete plantillas de artefactos a las que se refiere la Sección 3.1.

La documentación se publica como sitio de GitHub Pages, con traducciones del README al portugués, español, italiano y hebreo hechas con asistencia automática o de IA (docs/translations.md). Todo el código fuente —archivos de la *skill*, manifiestos del *plugin*, documentación, artefactos de prueba, código del *benchmark*— tiene licencia MIT; el texto del artículo, las figuras y los datos tienen licencia CC BY 4.0.

5. Observaciones preliminares

Sometimos a ASP a cinco evaluaciones de ingeniería estructuradas, que abarcan dominios heterogéneos. Esta sección separa tres cosas: lo que el autor observó en despachos reales de modelo (5.2), lo que el adjudicador determinista muestra sobre sí mismo con *fixtures* construidos (5.3) y lo que no se ha medido en absoluto.

5.1. El conjunto de evaluaciones

La Tabla 4 enumera las cinco evaluaciones de ingeniería.

Tabla 4
Las cinco evaluaciones de ingeniería

ID	Dominio	Tarea (abreviada)	Tamaño esperado	Aceptación
E1	Migración de esquema	Columna NOT NULL añadida a user_profiles, de 1M de filas	10	≥ 80%
E2	Despliegue en el edge	Función en el edge que llama a una API externa con límite de tasa	10	≥ 80%
E3	Refactorización	Utilidad de formateo de fechas usada en más de 30 archivos	10	≥ 80%
E4	Política RLS	Política de seguridad a nivel de fila que referencia una columna nueva	10	≥ 80%
E5	Planificación de tareas	Tarea cron cuyo *script* entra en conflicto con un servicio desactivado	10	≥ 80% y puntos 1 y 8 presentes

Cada evaluación especifica un INPUT (el enunciado de una tarea tal como un ingeniero sénior podría formulárselo a un colega), una lista EXPECTED de shadow points que un profesional competente sacaría a la luz y un umbral de ACCEPTANCE (por defecto 80%). E5 no es portable: dos de sus diez puntos esperados (1 y 8) dependen de la memoria privada de proyecto del autor y no pueden surgir solo a partir de su INPUT, de modo que mide la evocación de memoria tanto como la planificación. El texto completo de cada evaluación está en el repositorio complementario.

5.2. Lo que se observó en despachos reales (autor, no ciego)

El 2026-05-17 las evaluaciones se despacharon a subagentes nuevos: GREEN (con la disciplina de ASP inyectada: encuadre de pre-mortem, enumeración forzada de los 14 modos de MAST, exigencia de mitigación) en las cinco, y RED (a quien se pidió planificar como lo haría un ingeniero sénior, sin ningún *framework*) en tres, E1–E3; E4 y E5 se añadieron después y se ejecutaron solo en GREEN. Todos se ejecutaron con el modelo de sesión de la época (Claude Opus 4.7; el modelo no se registró por despacho), un despacho por condición. Los subagentes heredaron las instrucciones globales de agente del autor, de modo que la condición RED fue una línea base competente, no ingenua. El autor leyó cada plan frente a la lista esperada y registró un juicio de

cobertura; los resúmenes, con los identificadores de subagente, están versionados en tests/baseline-pressure-tests.md y tests/evals-summary.md. Las transcripciones en bruto no se versionaron y ya no están disponibles. Los *prompts* RED registrados allí son formulaciones en portugués, más breves, de E1–E3 que omiten detalles que las listas esperadas puntúan después: el *prompt* RED de E2 no indica el límite de tasa de 100 req/s de Resend, la API de correo a la que llama la tarea, y el de E1 no dice que el código de la aplicación ya referencia la nueva columna en cinco puntos de consulta. Parte del déficit de RED puede deberse, por tanto, a un artefacto del *prompt* y no a la ausencia del protocolo.

Bajo esa lectura no ciega, los tres planes RED cubrieron aproximadamente el 30–60% de sus listas esperadas (media cercana al 47%, contando las menciones parciales como cubiertas) y los planes GREEN cubrieron la totalidad o casi la totalidad de las suyas. Los resúmenes versionados, escritos en mayo de 2026, concluyen que ASP es "reproduciblemente mejor" (traducción nuestra); una nota sobre ellos retira esa conclusión, que no está respaldada. Estas cifras son un juicio del autor sobre texto no conservado, puntuado frente a listas escritas por el mismo autor, con el vocabulario de MAST que inyecta el *prompt* GREEN apareciendo en esas listas. Se informan como observación, no como medición, y ningún número de este párrafo lo verifica la capa de replicación.

5.3. Lo que el adjudicador determinista muestra sobre sí mismo

El paquete benchmark/ puntúa un plan frente a una lista esperada con un *matcher* determinista de 4 capas, la parte que decide si un plan menciona un punto esperado: nombre exacto, *sidecar* de sinónimos (una lista opcional de sinónimos aceptados), ventana deslizante de Jaccard (una puntuación de solapamiento de palabras sobre un tramo móvil del plan) y sin coincidencia. En la ejecución publicada, cada una de las 42 coincidencias procede de la primera capa (nombre exacto); no existe *sidecar* de sinónimos para ninguna tarea y la capa de Jaccard nunca se activa.

Su única ejecución publicada es una ejecución en **modo *mock***: los textos RED y GREEN son *fixtures* que el autor escribió para ejercitar el *pipeline* de extremo a extremo (benchmark/tests/fixtures/mock_runs/); con ellos, las capas 2 y 3 nunca producen una coincidencia (los solapamientos de la capa 3 se calculan y quedan por debajo del umbral), y solo las pruebas unitarias ejercitan una coincidencia a través de esas capas. Los propios *fixtures* se escribieron deliberadamente más débiles para RED y deliberadamente alineados con la lista esperada para GREEN. Con esos *fixtures*, el *matcher* puntúa RED en 4% y GREEN en 80% de media —una afirmación sobre cómo se escribieron los *fixtures*, no sobre ASP.

En símbolos, la capa 3 puntúa una ventana del plan frente a un punto esperado con la Ecuación 1, donde A es el conjunto de tokens distintos de la ventana y B el del punto esperado; las barras verticales cuentan los tokens de un conjunto. Una coincidencia exige una puntuación de al menos ,30.

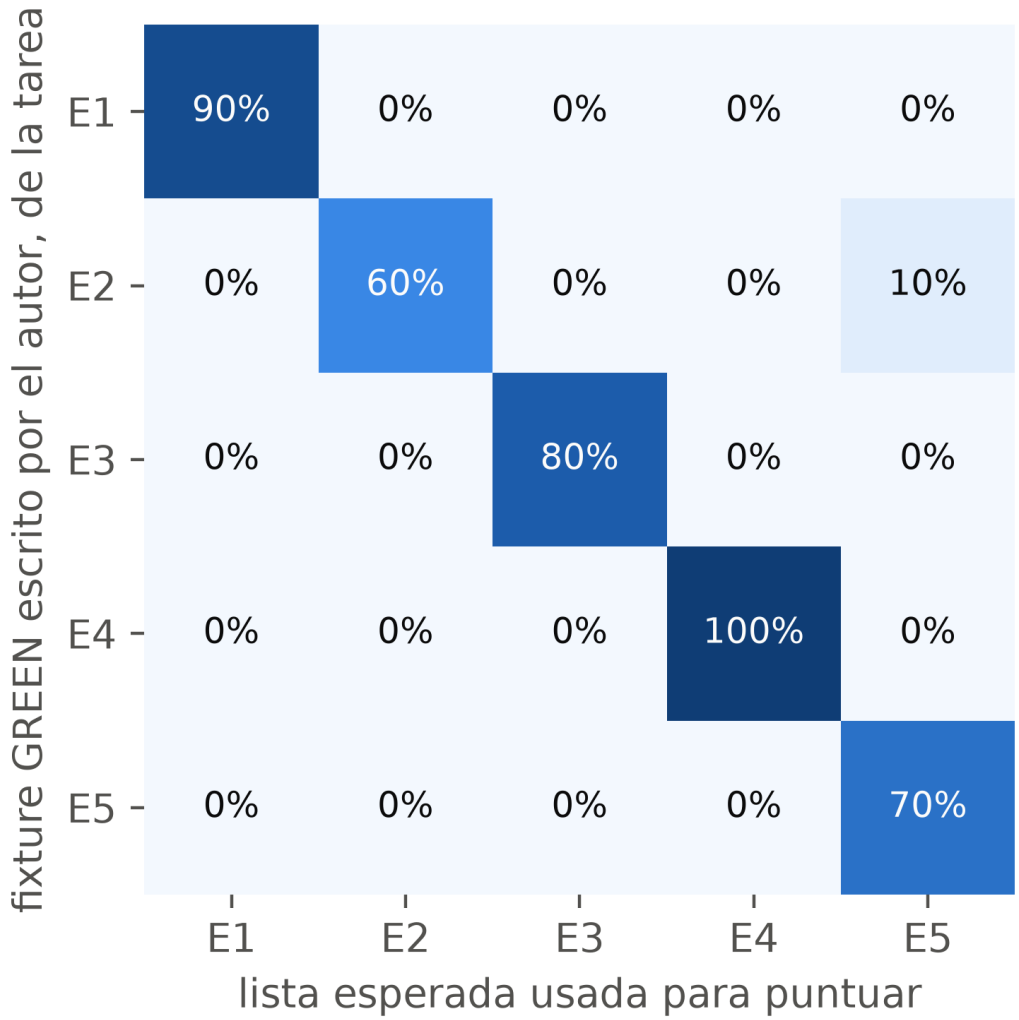
$$J = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

La propiedad útil que la ejecución *mock* sí establece es la **discriminación**: puntuar cada *fixture* GREEN frente a la lista esperada de una tarea *distinta* reduce la media del 80% al 0,5% (mayor celda fuera de tarea 10%; Figura 2). Es una evidencia más débil de lo que parece: el único acierto fuera de tarea es el nombre genérico de una palabra “Alerting”, y las demás celdas son cero porque la capa 1 exige que aparezca literalmente un nombre esperado completo. El *matcher* es además más estricto que los archivos de evaluación, que indican a un evaluador humano que acepte paráfrasis. La exhaustividad (*recall*) es otra cuestión, y los *fixtures* muestran un falso negativo: el *fixture* RED de E2 menciona un límite de tasa, reintentos y registro y aun así puntúa 0/10: su mejor solapamiento de ventana con el punto esperado “Rate-limit backoff” (su nombre y su descripción)

es un Jaccard de ,11, por debajo del umbral de ,30, y el *matcher* no hace *stemming* (“logging” no coincide con “logs”). La paráfrasis con bajo solapamiento léxico se pierde. La nueva puntuación expuso además un defecto, ya corregido: la ventana deslizante omitía los últimos tokens de un texto cuya longitud no coincidía con el paso; volver a puntuar todos los *fixtures* con la corrección no cambia ningún valor de cobertura, aunque el mejor solapamiento de tres puntos sin coincidencia sube ligeramente (E2 RED “Observability / logs” de ,00 a ,02), todavía muy por debajo del umbral. La capa 3 tiene además un techo aritmético: Jaccard divide por la unión, y una ventana de 40 tokens de los *fixtures* contiene de 27 a 40 palabras de contenido distintas (mediana 36); en cualquier ventana con 34 o más, un punto esperado con diez o menos tokens distintos no puede alcanzar ,30 aunque el plan los repita todos, y 19 de los 50 puntos esperados son así de cortos. El estudio de 2026-11 (Sección 5.5) sustituyó la capa 3 por el *recall* de tokens antes de que existiera ninguna lista de referencia. En las propias listas de los revisores, ni siquiera esa versión veía que dos revisores nombraban el mismo problema, así que un juez ciego pasó a ser la puntuación principal antes de recoger ningún dato; el *matcher* se informa como secundario.

Figura 2

Comprobación del instrumento: cada fixture GREEN construido por el autor puntuado frente a la lista esperada de cada tarea. No es un efecto de ASP



La Sección 5.5 informa de la medición; la Sección 6 enuncia las hipótesis de diseño a las que afecta.

5.4. El hallazgo sobre el código de salida de claude -p

Durante una batería de cuatro pruebas contra el ejecutor no interactivo claude -p, observamos que el binario devuelve el código de salida 0 incluso cuando el agente rechaza con cortesía el objetivo solicitado: el *harness* (el programa que ejecuta al agente) informa de un final de conversación

exitoso a pesar de que el agente rechaza semánticamente la tarea. El código de salida refleja el éxito a nivel de proceso del binario de la CLI, no el éxito semántico de la tarea del agente. Las Iron Laws 11 y 12 exigen --output-format json o stream-json y leer is_error, terminal_reason y stop_reason. Esos campos separan un corte por límite de turnos y un rechazo de clasificador de un turno completado, pero no un rechazo generado por el modelo de una entrega; por ello el ejecutor marca además las respuestas cuyo texto coincide con un patrón de rechazo, una heurística que el artículo complementario pone a prueba.

Este hallazgo es observacionalmente significativo para cualquier cadena de herramientas que envuelva claude -p en un *pipeline* de *shell*. Como la observación procedía de un único escenario de rechazo de nuestra propia batería, la **caracterizamos de forma independiente** en el artículo complementario Flores (2026b), que informa de una medición con N=50 en cuatro categorías de *prompts* que provocan rechazo (imposibles en un solo turno, de capacidad, de seguridad, de ambigüedad) en Claude Code 2.1.200 en julio de 2026. El modelo de sesión fue Fable 5; Opus 4.8 respondió 16 de los 50 *prompts*, incluidos los 12 de seguridad —de manera coherente con un traspaso automático (*fallback*), aunque los *envelopes* (el registro JSON que claude -p imprime al final de cada ejecución) registran qué modelo respondió, no por qué—. Veinticuatro ensayos terminaron con código de salida 0, todos con stop_reason: end_turn. El artículo complementario cuenta los rechazos entre ellos de cuatro maneras: 20 (40%, IC de Wilson al 95% 28–54%, un intervalo de confianza adecuado para muestras pequeñas) con la etiqueta no ciega del autor; 19 (38%) con la mayoría de un *codebook*, escrito después de que el autor leyera las respuestas y aplicado por cinco evaluadores LLM, cuatro de los cuales devolvieron etiquetas idénticas, todos del linaje del modelo bajo prueba, de modo que la mayoría es una comprobación del *codebook* y no una estimación de acuerdo entre evaluadores; 12 (24%) contando solo los rechazos de seguridad

y de comandos destructivos y excluyendo los ocho rechazos fundamentados de tareas imposibles en un solo turno; y 9 (18%) con el detector de frases precintado solo en inglés, que no capta las respuestas redactadas en portugués. Una instantánea anterior (mayo de 2026, Claude Code 2.1.143, Opus 4.7) dio 60% con el mismo detector, con un modelo y un idioma de respuesta distintos. Un segundo evaluador humano y una tercera instantánea bajo un preregistro redactado son los seguimientos pendientes.

5.5. Una medición prerregistrada (2026-11)

El estudio se prerregistró en el repositorio (benchmark/study-2026-11/PROTOCOL.md), y su registro encadenado de hashes recibió un sello de tiempo de OpenTimestamps, un servicio público de sellado de tiempo, antes de la primera ejecución. Sustituyó al Protocolo 2 de PREREGISTRATION.md, el plan anterior del repositorio para esta medición, que nunca se depositó. El diseño (Figura 3):

- **Tareas.** Seis peticiones de cambio en seis repositorios públicos, cada uno fijado en un *commit*: borrado lógico de usuarios (T1, plantilla FastAPI), un resumen semanal por correo (T2, Flask), vincular el inicio de sesión de Google con el de contraseña (T3, Django), reintentar POST por defecto (T4, la biblioteca HTTP ky), despliegues sin interrupción (T5, Django en Docker) y coautores por artículo (T6, Express con Prisma).
- **Condiciones.** Tres, cada una ejecutada cinco veces por tarea (90 ejecuciones) con claude -p en Claude Opus 5.5, en modo plan (un ajuste de Claude Code en el que el agente puede leer archivos pero no modificarlos), desde un clon nuevo y un entorno limpio, con el mismo contrato de salida: una tabla final de como máximo diez riesgos.
    - **simple:** “Plan this change.” (planifica este cambio)

    - **riesgos:** “Before planning, list the ways this change could go wrong.” (antes de planificar, enumera lo que puede salir mal)
    - **ASP:** /asp --standard, detenido tras pasar el plan por su única ronda de validador.

- **Listas de referencia.** Tres modelos de otros proveedores (Grok, GPT-OSS 120B, Gemini 3.8 Flash) leyeron cada repositorio y cada petición sin ver ASP y escribieron los riesgos que esperaban. Un juez ciego agrupó sus elementos, y un riesgo entró en la lista de referencia cuando al menos dos revisores lo nombraron (de 8 a 12 por tarea). Los modelos ocuparon el lugar de ingenieros independientes, lo que el protocolo declara como un debilitamiento.
- **Puntuación.** El juez principal (Gemini 3.8 Flash) vio solo las filas de la tabla de cada respuesta, con las palabras que revelarían la condición borradas, y marcó cada riesgo de referencia como cubierto o no. Un segundo juez (Grok) hizo lo mismo. Los dos coincidieron en el 91,9% de 855 decisiones por elemento (calculado a partir de benchmark/study-2026-11/judge/ y comprobado en tests/test_study_2026_11.py).

La Tabla 5 y la Figura 4 muestran la cobertura por condición.

Tabla 5
Cobertura de riesgos y tiempo por ejecución en el estudio de 2026-11, por condición

Condición	Cubiertos (juez principal)	Cubiertos (segundo juez)	Tiempo medio por ejecución
simple	56%	57%	58 s
riesgos	66%	70%	50 s
ASP	48%	54%	175 s

Figura 3

Diseño del estudio de 2026-11: los riesgos de referencia, escritos sin ver ASP, y las respuestas de las tres condiciones solo se encuentran en el juez ciego

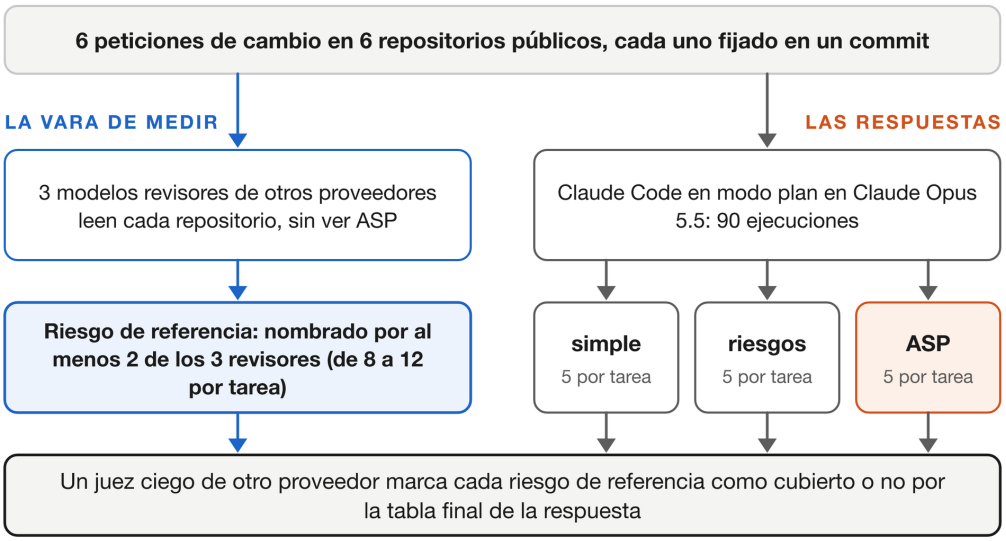
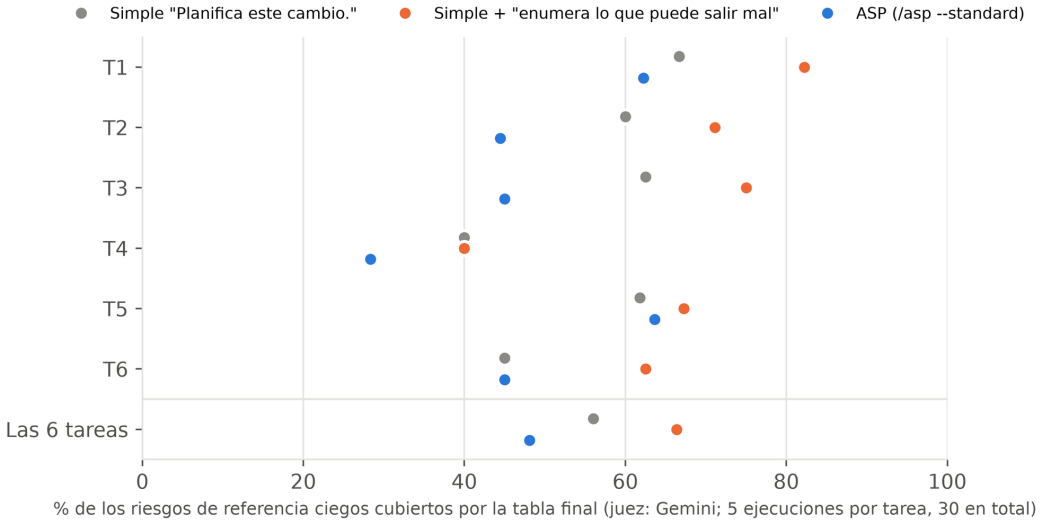


Figura 4

Fracción de los riesgos de referencia ciegos cubierta por la tabla final de cada condición, por tarea y en total (juez principal)



- **Primario.** ASP menos riesgos dio −,18 (juez principal) y −,16 (segundo juez). Fue negativo en las seis tareas, con una prueba de permutación unilateral, estratificada por tarea, p = 1,0 (con qué frecuencia barajar las etiquetas de condición dentro de cada tarea

da una diferencia igual de grande; la prueba pregunta si ASP supera a riesgos). La regla de decisión fijada de antemano da *ningún beneficio medible*.

- **Secundario, prerregistrado.** riesgos menos simple dio +,10 (p = ,001) y +,13 (p < ,001): pedir, en una frase, lo que puede salir mal aumentó la cobertura.
- **Exploratorio.** Las respuestas de ASP tratan algunos riesgos en sus supuestos y entregables y no en la tabla final. Cuando el juez principal leyó la respuesta completa, la diferencia se redujo a −,08 (66% frente a 74%), y ASP siguió por detrás en cinco de las seis tareas (veredictos en benchmark/study-2026-11/judge/gemini-full/, comprobados en tests/test_study_2026_11.py).
- **Límites.** Un modelo de sesión (en una de las 90 ejecuciones el *harness* llamó también a un modelo menor del mismo proveedor), seis tareas, listas de referencia y jueces que son todos modelos, y la cobertura como único resultado: no se midió si los riesgos enumerados son reales ni si llevan a mejor código.
- **Registro.** Cada ejecución, cada veredicto y el análisis están en el repositorio, encadenados por hash, y run_all.py vuelve a calcular cada número anterior.

6. Hipótesis de diseño

Destacamos cinco decisiones de diseño que creíamos fundamentales (*load-bearing*). Las cuatro primeras son hipótesis sobre el diseño de ASP, cada una enunciada con la observación que la motivó. El estudio de 2026-11 (Sección 5.5) midió una adaptación no interactiva de las Fases 0 a 6 en su conjunto y no encontró beneficio, lo que pesa en contra de ellas tomadas juntas, sin poner a prueba ninguna por separado. A las tres condiciones se les indicó que declararan sus supuestos

en lugar de preguntar, lo que sustituye las rondas de preguntas de ASP (Fases 0a y 0b); no se evaluó el uso interactivo. La quinta se apoya en la medición del artículo complementario.

Hipótesis: una taxonomía forzada saca a la luz más que un pre-mortem libre. En los borradores del autor, el pre-mortem libre se agrupaba en categorías de riesgo obvias (datos, integración) y subponderaba las menos salientes (hora y zona horaria, observabilidad, excelencia operativa, deriva de contratos). Ningún despacho puso a prueba el pre-mortem libre como condición (RED no tenía ningún *framework*), de modo que esto se apoya en la lectura que el autor hace de sus borradores. El *prompt* libre ("imagine the task has failed in 30 days; list 10+ causes") sería entonces necesario pero no suficiente, y el interrogatorio de los 14 modos de MAST sacaría a la luz categorías de las que la pasada libre se aleja agrupándose.

Hipótesis: el aislamiento de *prompts* del validador importa. Esperamos que los validadores que comparten contexto con los planificadores reproduzcan la confianza del planificador sobre el contenido compartido. La literatura convergente sobre los límites de la autocorrección intrínseca, el fallo de localización de errores y el sesgo de autopotenciación de los jueces apunta a ello; no lo hemos puesto a prueba dentro de ASP.

Hipótesis: los traspasos basados en archivos resisten mejor la presión de contexto que los basados en transcripciones. ASP persiste en disco el *charter*, el plan macro, el registro de entregables y el goal spec antes de que empiece la ejecución. En el modo opcional --goal, el hijo recibe solo el texto del Goal Spec por stdin (la entrada estándar) y devuelve un resultado JSON; el padre reconstruye el estado a partir de ese resultado y de los archivos en disco, de modo que ninguno de los dos depende de una transcripción que pueda haberse comprimido o truncado. No hemos comparado los dos traspasos.

Hipótesis: las restricciones enunciadas (Iron Laws) hacen más difícil omitir una *skill*. En nuestra experiencia, las *skills* que se leen como "pautas" a menudo se omiten bajo presión de contexto. Las *skills* que emiten una lista de innegociables, con plantillas cuyos campos deben rellenarse antes de la siguiente fase, son en nuestro uso más difíciles de omitir. El cumplimiento no se ha medido.

Medido en otro lugar: tratar los códigos de salida de claude -p como señales de éxito semántico es una trampa silenciosa. Este es el hallazgo informado en Flores (2026b); lo mencionamos aquí porque la implicación de diseño (Iron Laws 11 y 12) fue importante para la fiabilidad del sistema.

7. Amenazas a la validez

Tratamos las amenazas de forma explícita en las cuatro categorías habituales en los informes empíricos de ingeniería de software.

Validez de constructo. En las evaluaciones de las Secciones 5.1–5.2, el mismo investigador que diseñó el protocolo redactó las listas de shadow points esperados *y* adjudicó a mano la cobertura. Incluso con una rúbrica escrita, esta es la mayor amenaza individual para la interpretabilidad de la observación. El adjudicador determinista que distribuimos la mitigaría una vez aplicado a transcripciones reales y conservadas; hasta ahora solo se ha ejecutado sobre *fixtures* construidos (§5.3), y no aborda la cuestión previa de si las propias listas esperadas están sesgadas hacia lo que ASP saca a la luz. **La redacción ciega de nuevo por revisores independientes es el seguimiento más sólido.** El estudio de la Sección 5.5 tuvo sus listas de referencia redactadas a ciegas, por tres modelos de otros proveedores en lugar de ingenieros, para seis tareas nuevas y no para estas cinco.

Validez interna. Con un despacho por celda, GREEN en cinco tareas y RED en tres, las evaluaciones de las Secciones 5.1–5.2 a lo sumo sugieren que el protocolo puede cambiar las transcripciones en estos casos; *no* bastan para estimar un tamaño del efecto, establecer causalidad ni respaldar inferencia estadística, y no se informa de ninguna prueba estadística para ellas. No se ha realizado ninguna ablación formal de componentes (la atribución cualitativa de carácter fundamental a la lista de verificación MAST del §6 es una hipótesis). El umbral de aceptación del 80% es elección del autor y no se deriva de la literatura previa.

Validez externa. Para las evaluaciones de las Secciones 5.1–5.2: un único modelo subyacente (Claude Opus 4.7, presumido a partir de la sesión; no registrado por despacho), un único entorno (la estación de trabajo macOS del autor y la versión de Claude Code de mayo de 2026, no registrada por despacho), cinco tareas de ingeniería extraídas de la experiencia del autor en su dominio. No afirmamos ninguna generalización más allá de estas condiciones.

Fiabilidad. Todos los enunciados INPUT de las evaluaciones, las listas de shadow points esperados, los *fixtures* de la ejecución *mock* y el código del adjudicador determinista están versionados en el repositorio complementario; las transcripciones de los despachos del 2026-05-17 no se conservaron, de modo que las observaciones del §5.2 no pueden volver a comprobarse. El paquete de *benchmark* construye un manifiesto encadenado criptográficamente por ejecución, de modo que cualquier byte editado en los archivos que el manifiesto cubre (entorno, código, entradas, salidas y adjudicación) invalida el hash del manifiesto; el propio archivo de manifiesto y el report.md de la ejecución quedan fuera de él.

El estudio de la Sección 5.5. Externa: un modelo (Claude Opus 5.5) y seis tareas. De constructo: las listas de referencia y los jueces son modelos, en lugar de ingenieros independientes, lo que el protocolo declara como un debilitamiento; la cobertura es el único resultado, de modo que no se

midió si los riesgos enumerados son reales ni si llevan a mejor código; y el contrato de la tabla final puede dejar sin contar riesgos que ASP trata en sus supuestos y entregables (leída la respuesta completa, la diferencia se redujo, pero no se cerró). Interna: el estudio midió las fases de planificación del protocolo como un todo, no cada componente (Sección 6). A los revisores que escribieron las listas de referencia se les preguntó por "the concrete ways this change could go wrong" (las formas concretas en que este cambio podría salir mal), palabras próximas a las de la condición riesgos, lo que puede favorecer a esa condición en la cobertura. El juez principal también escribió una de las tres listas de los revisores, y 50 de los 57 riesgos de referencia llevan su redacción; el segundo juez escribió otra lista, de modo que ninguno de los dos jueces es independiente de la vara que aplica.

8. Limitaciones y trabajo futuro

Las limitaciones de este informe de experiencia se corresponden una a una con las amenazas anteriores. El primer seguimiento nombrado en versiones anteriores, un estudio confirmatorio del pre-mortem de una frase (la petición de la condición riesgos, que conserva la parte de enumeración de un pre-mortem y no pide al modelo que imagine que el cambio ya ha fracasado), se presenta ahora en Flores (2026c). En doce tareas fijadas y dos modelos de un mismo proveedor, Claude Opus 5.5 y Claude Sonnet 5.5, la frase elevó la cobertura sobre una petición simple de planificación del 46,1% al 57,5% en 240 ejecuciones ($p < ,001$); esa afirmación se limita a listas de referencia escritas por modelos y a planes, no a código. Los seguimientos de mayor prioridad que quedan, por orden, son:

1. **Ablación formal**, ahora que el estudio confirmatorio se ha completado: antes de probar qué componente del protocolo ayuda (lista MAST, aislamiento del validador, *intake* de la Fase 0a), probar si alguno supera a la frase única.

2. **Un ingeniero humano en una muestra de las listas de referencia**, junto a las listas escritas por modelos de la Sección 5.5 y de Flores (2026c), y, para el estudio de la Sección 5.5, una medida de precisión (¿son reales los riesgos enumerados?) junto a la cobertura.
3. **Un segundo evaluador humano y una tercera instantánea de claude -p** para el hallazgo complementario (Flores, 2026b, §8), esta última bajo el Protocolo 1 de PREREGISTRATION.md.
4. **Evaluación entre modelos.** Volver a ejecutar en Claude Sonnet, GPT-5, Gemini 2.5 y un modelo de pesos abiertos.
5. **Comparación directa con PreFlect.** Ejecutar ASP y una reimplementación de PreFlect sobre el mismo conjunto de tareas.
6. **Adjudicador basado en *embeddings*** como v0.2 del *benchmark*, con un modelo sentence-transformer fijado, para elevar el piso de exhaustividad en paráfrasis léxicamente disjuntas.
7. **Replicación en *benchmarks* estandarizados.** El conjunto de cinco tareas no es un *benchmark* público de agentes; replicar la dirección en SWE-bench (Jimenez et al., 2024) o en una suite estandarizada comparable pondría a prueba si sobrevive fuera de las tareas elegidas por el autor.

9. Conclusión

Describimos ASP, una *skill* operativa de pre-mortem para agentes de programación con LLM, que integra el hallazgo de retrospectiva prospectiva de Mitchell, Russo y Pennington (1989), la operacionalización del pre-mortem de Klein (2007) y la taxonomía de fallos de 14 modos MAST

de Cemri et al. (2025) como disciplina de tiempo de ejecución bajo un contrato de doce Iron Laws. La *skill* se distribuye como código abierto con tres vías de instalación. La observación inicial del autor, no ciega, de que los planes asistidos cubrían más modos de fallo no sobrevivió a la medición. En un estudio prerregistrado en seis repositorios reales, las fases de planificación del protocolo cubrieron menos riesgos de los revisores que Claude Code simple al que se pidió, en una frase, enumerar lo que puede salir mal, y esa frase sola superó a una petición simple de planificación. Desde entonces, un estudio confirmatorio ha probado esa comparación como pregunta principal (Flores, 2026c).

Observamos además que claude -p devuelve el código de salida 0 incluso cuando el agente rechaza con cortesía el objetivo solicitado. El proveedor documenta el comportamiento para su API, como cataloga el artículo complementario (Flores, 2026b, §2); ese artículo añade un recuento en la frontera de la CLI.

Con el estudio confirmatorio del pre-mortem de una frase ya presentado (Flores, 2026c), el seguimiento más fuerte para el propio ASP es la ablación formal de la Sección 8. El trabajo complementario más útil es un segundo evaluador humano y una tercera instantánea preregistrada de la medición de claude -p (Flores, 2026b).

10. Declaraciones

Conflicto de intereses. El autor creó y mantiene ASP, el *plugin* que este informe describe, y dirige el laboratorio que lo publica. El estudio prerregistrado de la Sección 5.5 no halló beneficio medible del protocolo.

Financiación. Este trabajo no recibió financiación externa. Los costos de uso de los modelos los pagó el autor.

Contribución del autor. Autor único: concepción, diseño, recogida de datos, análisis, redacción y revisión.

Uso de herramientas de IA. Los modelos de lenguaje son el objeto de este informe y tienen tres papeles en el estudio de la Sección 5.5: los agentes probados, los redactores de las listas de referencia y los jueces. Un asistente de IA se usó también en la programación, en el análisis y en la redacción y traducción del texto, bajo la dirección del autor, como detallan los Agradecimientos. El autor comprobó cada número con los datos y responde por el contenido.

Agradecimientos

Este informe de experiencia se preparó en colaboración explícita con un humano en el bucle y con un asistente de IA, usado como socio de redacción, verificación y revisión. Las decisiones de metodología, el diseño empírico y la finalización las realizó el autor humano. El modelo de colaboración está documentado en las Partes 5–6 de docs/ARCHITECTURE.md del repositorio complementario.

El autor agradece a los mantenedores del *framework* de *skills* obra/superpowers, a los colaboradores del estándar abierto agentskills.io y al equipo de Claude Code de Anthropic por la primitiva /goal, cuyo comportamiento empírico motivó las Iron Laws 11 y 12.

Referencias

Anthropic. (2026a). *Plugins reference* [Software documentation]. https://code.claude.com/docs/en/plugins-reference

Anthropic. (2026b). *Agent Skills best practices* [Software documentation]. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, *1*(1), 11–33. https://doi.org/10.1109/TDSC.2004.2

Barke, S., Goyal, A., Khare, A., Singh, A., Nath, S., & Bansal, C. (2026). *AgentRx: Diagnosing AI agent failures from execution trajectories*. arXiv:2602.02475. https://arxiv.org/abs/2602.02475

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., & Stoica, I. (2025). Why do multi-agent LLM systems fail? *Advances in Neural Information Processing Systems, 38*, 135676–135729. https://doi.org/10.52202/085713-4082

Deshpande, D., Gangal, V., Mehta, H., Krishnan, J., Kannappan, A., & Qian, R. (2025). *TRAIL: Trace reasoning and agentic issue localization*. arXiv:2505.08638. https://arxiv.org/abs/2505.08638

Erdogan, L. E., Lee, N., Kim, S., Moon, S., Furuta, H., Anumanchipalli, G., Keutzer, K., & Gholami, A. (2025). Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning* (Vol. 267, pp. 15419–15462). PMLR. https://proceedings.mlr.press/v267/erdogan25a.html

Flores, C. U. (2026a). *ASP — Anticipating Shadow Points: Claude Code plugin and replication package* (Version 2.0.0) [Computer software]. GitHub. https://github.com/ulissesflores/anticipating-shadow-points https://doi.org/10.5281/zenodo.20276631

Flores, C. U. (2026b). *Graceful refusals as unsignalled failures: Characterising claude -p exit-code semantics on agent declines* (Version 2.4) [Technical note]. Zenodo. https://doi.org/10.5281/zenodo.24065934

Flores, C. U. (2026c). *One sentence before the plan: A pre-registered confirmatory study of pre-mortem prompting in LLM coding agents* (Version 1.0) [Technical report]. Zenodo. https://doi.org/10.5281/zenodo.24410437

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). *Large language models cannot self-correct reasoning yet*. ICLR 2024. https://arxiv.org/abs/2310.01798

IEEE. (2017). *IEEE standard for system, software, and hardware verification and validation* (IEEE Std 1012-2016). https://doi.org/10.1109/IEEESTD.2017.8055462

International Electrotechnical Commission. (2016). *Hazard and operability studies (HAZOP studies) — Application guide* (IEC 61882:2016, Edition 2.0). https://webstore.iec.ch/en/publication/24321

International Electrotechnical Commission. (2018). *Failure modes and effects analysis (FMEA and FMECA)* (IEC 60812:2018). https://webstore.iec.ch/en/publication/26359

International Electrotechnical Commission. (2019). *Risk management — Risk assessment techniques* (IEC 31010:2019, Edition 2.0). https://webstore.iec.ch/en/publication/59809

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). *SWE-bench: Can language models resolve real-world GitHub issues?* ICLR 2024. https://arxiv.org/abs/2310.06770

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L. P., & Murthy, A. B. (2024). Position: LLMs can't plan, but can help planning in LLM-Modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning* (Vol. 235, pp. 22895–22907). PMLR. https://proceedings.mlr.press/v235/kambhampati24a.html

Kamoi, R., Zhang, Y., Zhang, N., Han, J., & Zhang, R. (2024). When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics, 12*, 1417–1440. https://doi.org/10.1162/tacl_a_00713

Klein, G. (2007). Performing a project premortem. *Harvard Business Review*, *85*(9), 18–19.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., & Clark, P. (2023). *Self-Refine: Iterative refinement with self-feedback*. NeurIPS 2023. https://arxiv.org/abs/2303.17651

Meyer, B. (1992). Applying "design by contract." *Computer*, *25*(10), 40–51. https://doi.org/10.1109/2.161279

Mitchell, D. J., Russo, J. E., & Pennington, N. (1989). Back to the future: Temporal perspective in the explanation of events. *Journal of Behavioral Decision Making*, *2*(1), 25–38. https://doi.org/10.1002/bdm.3960020103

Popper, K. R. (2002). *The logic of scientific discovery*. Routledge. (Original work published 1959)

Shingo, S. (1986). *Zero quality control: Source inspection and the poka-yoke system* (A. P. Dillon, Trans.). Productivity Press. (Original work published 1985)

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems, 36*, 8634–8652. https://doi.org/10.52202/075280-0377

Tyen, G., Mansoor, H., Cărbune, V., Chen, P., & Mak, T. (2024). LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*. https://arxiv.org/abs/2311.08516

U.S. Department of Defense. (1980). *Procedures for performing a failure mode, effects and criticality analysis* (MIL-STD-1629A). https://everyspec.com/MIL-STD/MIL-STD-1600-1699/MIL_STD_1629A_1556/

Wang, H., Cao, Y., Lin, L., & Chen, J. (2026). *PreFlect: From retrospective to prospective reflection in large language model agents*. arXiv. https://arxiv.org/abs/2602.07187

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). *AutoGen: Enabling next-gen LLM applications via multi-agent conversation*. arXiv. https://arxiv.org/abs/2308.08155

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing reasoning and acting in language models*. ICLR 2023. https://arxiv.org/abs/2210.03629

Yuan, D., Luo, Y., Zhuang, X., Rodrigues, G. R., Zhao, X., Zhang, Y., Jain, P. U., & Stumm, M. (2014). Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)* (pp. 249–265). USENIX Association. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

Zhang, G., Wang, J., Chen, J., Zhou, W., Wang, K., & Yan, S. (2025). *AgenTracer: Who is inducing failure in the LLM agentic systems?* arXiv:2509.03312. https://arxiv.org/abs/2509.03312

Zhang, J., Peng, X., Chen, Q., Li, Y., Hayashi, H., & Wu, C.-S. (2026). *Prospective hindsight: Self-calibrating reinforcement learning via prediction-reality gaps*. arXiv:2610.02740. https://arxiv.org/abs/2610.02740

Zhang, S., Yin, M., Zhang, J., Liu, J., Han, Z., Zhang, J., Li, B., Wang, C., Wang, H., Chen, Y., & Wu, Q. (2025). Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning* (Vol. 267, pp. 76583–76599). PMLR. https://proceedings.mlr.press/v267/zhang25cq.html

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. NeurIPS 2023 D&B. https://arxiv.org/abs/2306.05685

Zhu, K., Liu, Z., Li, B., Tian, M., Yang, Y., Zhang, J., Han, P., Xie, Q., Cui, F., Zhang, W., Ma, X., Yu, X., Ramesh, G., Wu, J., Liu, Z., Lu, P., Zou, J., & You, J. (2025). *Where LLM agents fail and how they can learn from failures*. arXiv. https://arxiv.org/abs/2509.25370

Apéndice A — Estado de la verificación de citas

Toda afirmación del cuerpo cita una fuente primaria cuando existe. Los párrafos siguientes registran cuándo se comprobó cada grupo de referencias.

Las referencias presentes en la versión 1.0 se verificaron mediante consulta directa de la URL el 2026-05-17; las referencias añadidas en la revisión de julio de 2026 (Flores, 2026b; las entradas del linaje clásico de la Sección 2.5) se comprobaron al añadirse; y las entradas añadidas en noviembre de 2026 (Barke et al., 2026; Deshpande et al., 2025; G. Zhang et al., 2025; J. Zhang et al., 2026) se comprobaron frente a sus registros de arXiv, y Cemri et al. (2025) frente a su registro en las actas de NeurIPS. En el mismo mes, Kambhampati et al. (2024) y Shinn et al. (2023) se comprobaron frente a sus registros en las actas (PMLR y NeurIPS), y sus entradas ahora citan esos registros. Flores (2026c), añadido también en noviembre de 2026, se comprobó frente a su manuscrito en el repositorio complementario (paper/one-sentence-premortem.md). Los nombres de los 14 modos de MAST se citan literalmente de Cemri et al. (2025) y se listan en el archivo de *skill* complementario plugins/anticipating-shadow-points/skills/anticipating-shadow-points/mast-checklist.md. Mitchell et al. (1989) es el estudio primario de la retrospectiva prospectiva; la cifra de ≈30% es el resumen que hace Klein (2007) de aquel, que no pudimos rederivar a partir del artículo de 1989, y Klein (2007) es además la receta operativa.

Las nueve referencias introducidas en el §2.5 (Avizienis et al., 2004; International Electrotechnical Commission [IEC 60812:2018], 2018; International Electrotechnical Commission [IEC 61882:2016], 2016, comprobada frente al catálogo de la IEC el 2026-11-07; IEEE [Std 1012-2016], 2017; Meyer, 1992; Popper, 1959/2002; Shingo, 1986; U.S. Department of Defense [MIL-STD-1629A], 1980; Yuan et al., 2014) se verificaron frente a su fuente primaria —DOI, ISBN o entrada de catálogo de la norma— el 2026-07-08.

Apéndice B — Conjunto de evaluaciones y reproducibilidad

Las cinco evaluaciones de ingeniería están versionadas en el repositorio complementario, en la carpeta plugins/anticipating-shadow-points/skills/anticipating-shadow-points/evals/:

- 01-supabase-migration.md (E1)
- 02-edge-function-deploy.md (E2)
- 03-refactor-shared-util.md (E3)
- 04-rls-policy-change.md (E4)
- 05-cron-skill-conflict.md (E5)

Cada archivo contiene el enunciado INPUT, la lista de shadow points EXPECTED y el umbral de ACCEPTANCE, todos redactados por el diseñador del protocolo. El paquete de *benchmark* en benchmark/ proporciona:

- un adjudicador determinista de 4 capas (exacto / sinónimo / Jaccard / sin coincidencia);
- procedencia criptográfica por ejecución (entorno, código, entradas, salidas y adjudicación con hash en una cadena de tipo Merkle);
- un subcomando verify que recorre de nuevo un directorio de ejecución e informa de cualquier deriva a nivel de byte en los archivos que el manifiesto cubre;
- 31 pruebas unitarias que cubren el adjudicador, los analizadores de evaluaciones y de respuestas, el manifiesto y el hasher, incluido el contrato de determinismo.

Reproducir los números calculados de las Secciones 5.3–5.5 (sin llamadas a modelos). La Tabla B1 enumera lo que el repositorio contiene para ellos.

Tabla B1

Lo que el repositorio contiene para las Secciones 5.3 a 5.5

Material	**Dónde en el repositorio**	**Qué puede comprobar el lector con él**
Las cinco evaluaciones	plugins/anticipating-shadow-points/skills/ anticipating-shadow-points/evals/	Cada tarea, su lista esperada y su regla de aceptación (Tabla 4)
Los planes de ejemplo del autor (RED y GREEN) y la ejecución que los puntuó	benchmark/tests/fixtures/mock_runs/ y benchmark/runs/published/	Los planes que puntuó el *matcher* y las puntuaciones detrás de la Figura 2
La ejecución de julio de 2026 de claude -p de la nota complementaria	data/spike-2026-07-08/	Los recuentos citados en la Sección 5.4
El estudio de 2026-11	benchmark/study-2026-11/	Protocolo, tareas, listas de referencia, 90 ejecuciones, veredictos de ambos jueces y registro encadenado de hashes, detrás de la Sección 5.5 y la Figura 4
El cálculo	analysis/results.py y benchmark/ study-2026-11/analyze.py	Cómo se calcula cada número de las Secciones 5.3 a 5.5
Las comprobaciones	tests/test_paper_numbers.py y tests/test_ study_2026_11.py	Que cada uno de esos números coincide con los datos, y que la ejecución puntuada está etiquetada como mock
El sello	output/hash-chain.md	Una única huella SHA-256 sobre código, datos y resultados, de modo que cualquier cambio posterior se note

Una huella SHA-256 es un código de 64 caracteres calculado a partir de un archivo; cambiar un solo byte del archivo cambia el código. Un único comando, python run_all.py, ejecutado dentro de la carpeta del repositorio, rehace todo lo de la Tabla B1 y no llama a ningún modelo; requiere Python 3.11 o posterior, y el README del repositorio enumera los cuatro comandos que descargan el repositorio, instalan lo que necesita y lo ejecutan. Las observaciones del autor de la Sección 5.2 no pueden reproducirse: las salidas del modelo a partir de las cuales se leyeron no se conservaron.

Recoger evidencia real (cuesta llamadas a modelos). Instale el paquete de *benchmark* y ejecute asp-benchmark run --mode real --model <model> seguido de asp-benchmark verify <run-id>. El *runner* de este paquete no se usó para ningún número publicado. El estudio de 2026-11 de la Sección 5.5 tiene su propio *runner*, juez y registro encadenado de hashes en benchmark/study-2026-11/, y python run_all.py vuelve a calcular los números de ese estudio a partir de los veredictos versionados, sin llamar a un modelo. El protocolo se especifica de forma independiente de la implementación en Python en benchmark/PROTOCOL.md, de modo que pueda reimplementarse en otro lenguaje.

Nota Técnica

Versión 2.4 · noviembre de 2026 · publicado por primera vez el 18 de mayo de 2026

Serie Codex Hash Research Report 2026-01

Licencia Texto, figuras y datos del artículo: CC BY 4.0. Software complementario: MIT.

El proyecto ASP. ASP (Anticipating Shadow Points) es un *plugin* de código abierto para Claude Code, el agente de programación de línea de comandos de Anthropic, que hace al agente enumerar lo que podría salir mal antes de escribir código. El *plugin* y el material de replicación están en un repositorio público (Flores, 2026a); este informe de experiencia lo describe y una nota técnica aparte mide una de sus reglas: no confiar nunca en el código de salida de claude -p (Flores, 2026b). La Sección 5.5 trae la primera medición de si ASP mejora los planes de un agente; un tercer artículo trae el estudio confirmatorio del pre-mortem de una frase al que apuntó (Flores, 2026c).

DOI. Un DOI (Digital Object Identifier) es un enlace permanente a una obra depositada. Zenodo, archivo público de investigación gestionado por el CERN, guarda este informe de experiencia como registro propio: https://doi.org/10.5281/zenodo.24065968, que abre siempre la versión más reciente; esta versión, la 2.4, es https://doi.org/10.5281/zenodo.24410420. El *plugin*, los datos y el código forman otro registro (Flores, 2026a), cuyo DOI de concepto, https://doi.org/10.5281/zenodo.20276631, abre siempre la *release* más reciente; a este informe de experiencia corresponde la 2.0.0. Las de mayo de 2026 (1.0.0 a 1.0.3) presentaban las puntuaciones que el *matcher* daba a los planes de ejemplo del autor como lectura de salidas reales del modelo; la 2.0.0 las sustituye.

Correcciones. La versión 1.0 de este informe también presentaba las cinco evaluaciones como medidas y confundía tres cosas que la Sección 5 ahora separa: lo que el autor observó en despachos

reales, lo que el adjudicador muestra sobre sí mismo con *fixtures* construidos y lo que no se ha medido. La versión 2.0 lo corrigió, añadió sobre los resúmenes versionados de mayo la nota que retira su conclusión de que ASP es “reproduciblemente mejor” (Sección 5.2) y corrigió el defecto de la ventana deslizante de la Sección 5.3.

Traducciones y envío. Las traducciones de la documentación se describen en la Sección 4; los planes de envío dependen de los seguimientos de la Sección 8.

Replicación. Todo número calculado en las Secciones 5.3 a 5.5 puede recalcularse desde el repositorio sin llamar a ningún modelo; el Apéndice B dice qué hay y cómo ejecutarlo.

Sobre el autor

Carlos Ulisses Flores es CTO e Investigador Jefe de Codex Hash Research Laboratory, un laboratorio independiente de I+D en São Paulo, Brasil. Construye sistemas de producción donde el fallo silencioso sale caro —históricamente en trading algorítmico, IoT embebido y criptografía en el borde, y hoy en herramientas de IA agéntica—. La *skill* ASP descrita en este informe de experiencia surgió de esta última línea de trabajo; el hallazgo empírico informado en la nota técnica complementaria surgió de usar el ejecutor claude -p de Anthropic como núcleo de ejecución autónoma y de verlo devolver el código de salida 0 ante un rechazo cortés del agente.

Escribe en portugués, inglés, español e italiano.

Identificadores · ORCID: 0000-0002-6034-7765 · Lattes (registro nacional de currículos de investigadores de Brasil): 6905246706890561

Contacto · c.ulisses@gmail.com · https://ulissesflores.com